# BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

## IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images, please do not report the images to the Image Problems Mailbox.**

(12) UK Patent Application (19) GB (11) 2 331 605 (13) A

(43) Date of A Publication 26.05.1999

(21) Application No 9821777.1

(22) Date of Filing 06.10.1998

(30) Priority Data
(31) 08946323 (32) 07.10.1997 (33) US

(51) INT CL[6]
G06F 9/44 9/45

(52) UK CL (Edition Q )
G4A APL AUB

(56) Documents Cited
Picking out packets for perusal, Review of EtherPeek at www.zdnet.com, 6 January 1997 Review of Eagle 3.1 at www.zdnet.com, taken from ZD Internet Magazine February 1997 NetSense ProAnalyst review at www.softseek.com and at www.net3group.com, 8 July 1997 Anasil review at www.softseek.com and at www.lfnetworks.com, 30 June 1998

(58) Field of Search
UK CL (Edition Q ) G4A AKS APL AUB
INT CL[6] G06F 9/44 9/45 17/50
Online: EPODOC INTERNET PAJ WPI

(71) Applicant(s)
3COM Corporation
(Incorporated in USA - California)
5400 Bayfront Plaza, P.O.Box 58145, Santa Clara, California 95052-8145, United States of America

(72) Inventor(s)
Jing Xiang
Darryl Black

(74) Agent and/or Address for Service
Langner Parry
High Holborn House, 52-54 High Holborn, LONDON, WC1V 6RR, United Kingdom

(54) Abstract Title
**Packet filter compiler program**

(57) Disclosed is a computer program, driven by a graphical user interface (GUI), for building packet filters for use in a networked environment. The graphical user interface is adapted to assist a user in the construction and design of a packet filter such that the user is able to build a packet filter without being extensively trained in all of the numerous parameters involved in packet filter construction: this may be through the use of a wizard (1100) for inexperienced users or via a calculator type interface (1200) for more experienced users. In one embodiment, the program receives expression-based statements from the user. These expression-based statements indicate the design characteristics of the packet filter which the user wishes to build. The program compiles the expression-based statements into a language which is suitable for the formation of the packet filter. Finally the packet filter builder creates the packet filter desired by the user by combining a sequence of compiler statements.

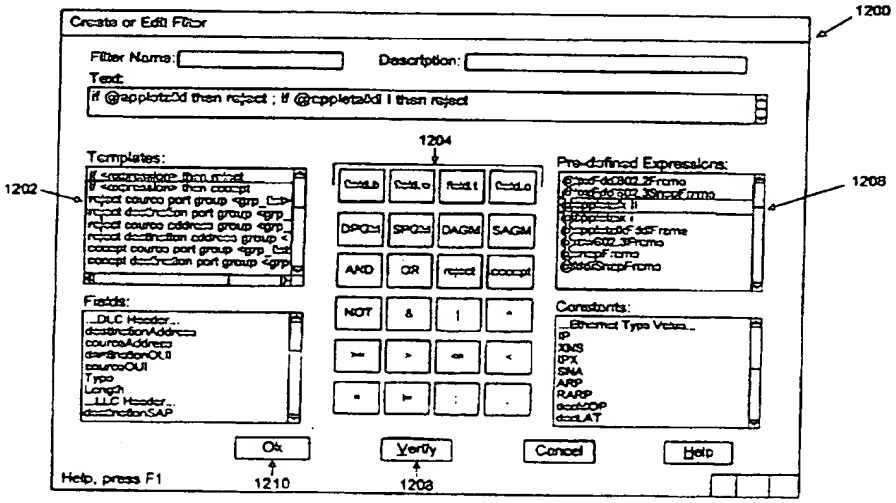



FIG. 10



GB 2 331 605 A

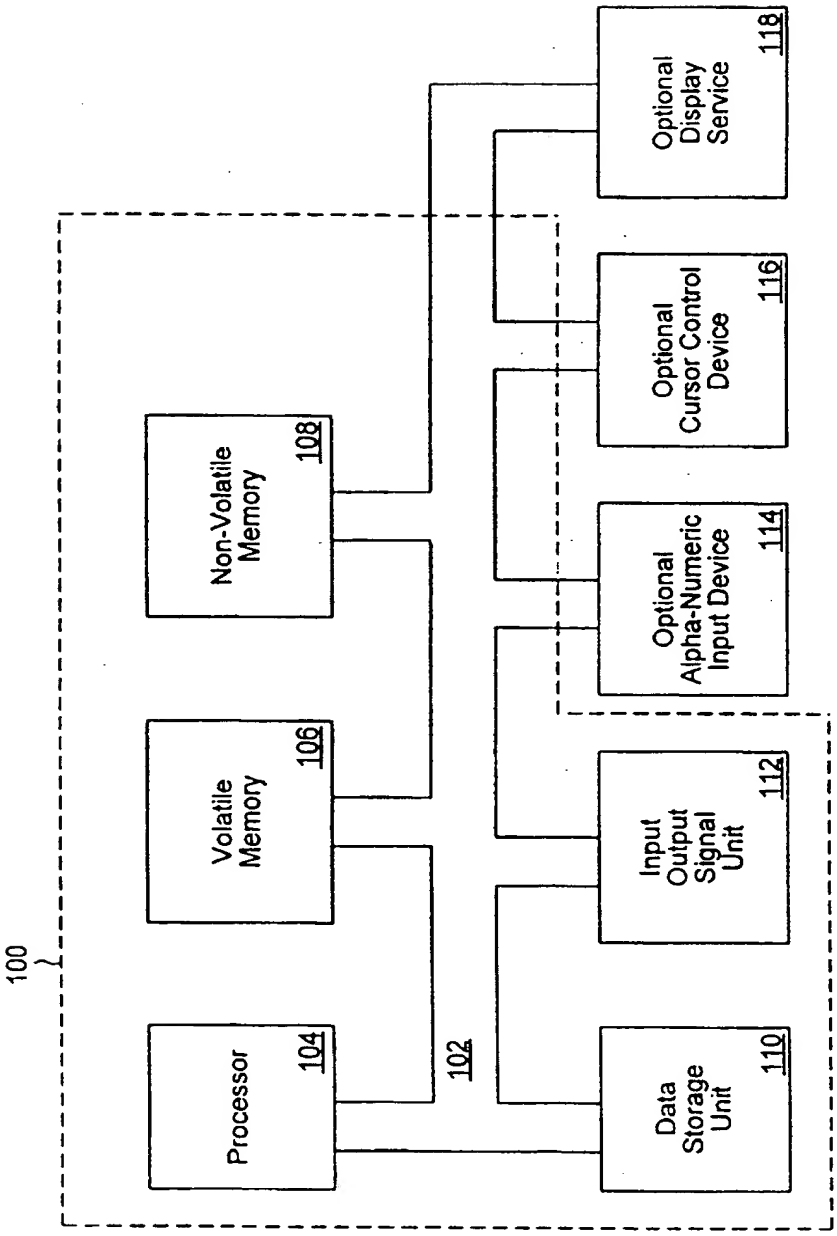

100
Processor
104
Volatile Memory
106
Non-Volatile Memory
108
102
Data Storage Unit
110
Input Output Signal Unit
112
Optional Alpha-Numeric Input Device
114
Optional Cursor Control Device
116
Optional Display Service
118

FIG. 1

200

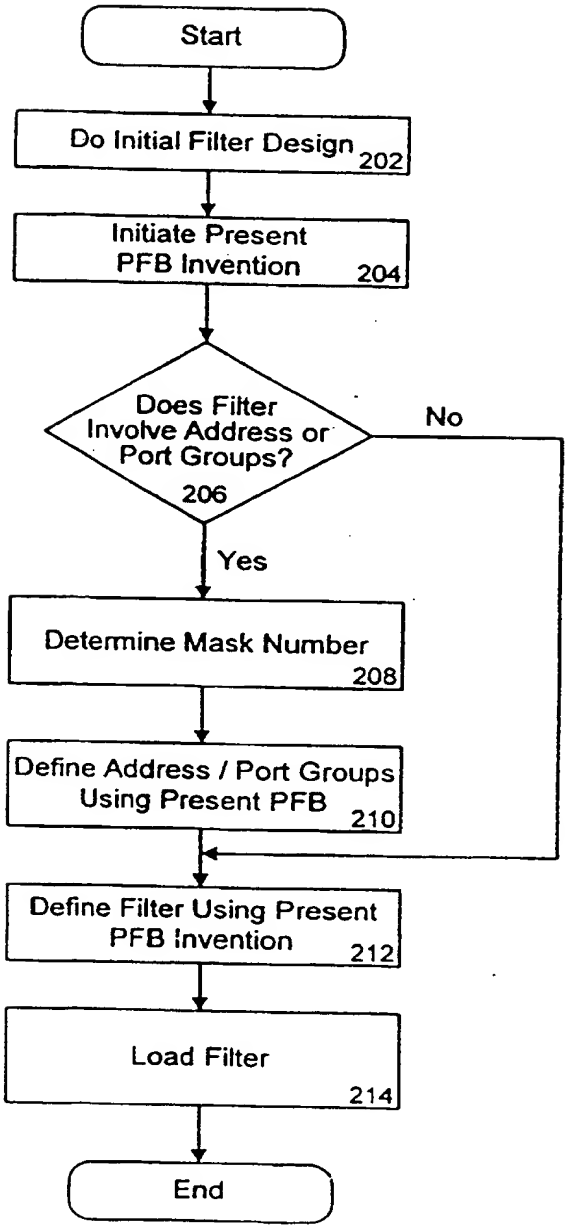

Start
Do Initial Filter Design 202
Initiate Present PFB Invention 204
Does Filter Involve Address or Port Groups? 206
No
Yes
Determine Mask Number 208
Define Address / Port Groups Using Present PFB 210
Define Filter Using Present PFB Invention 212
Load Filter 214
End

FIG. 2

300

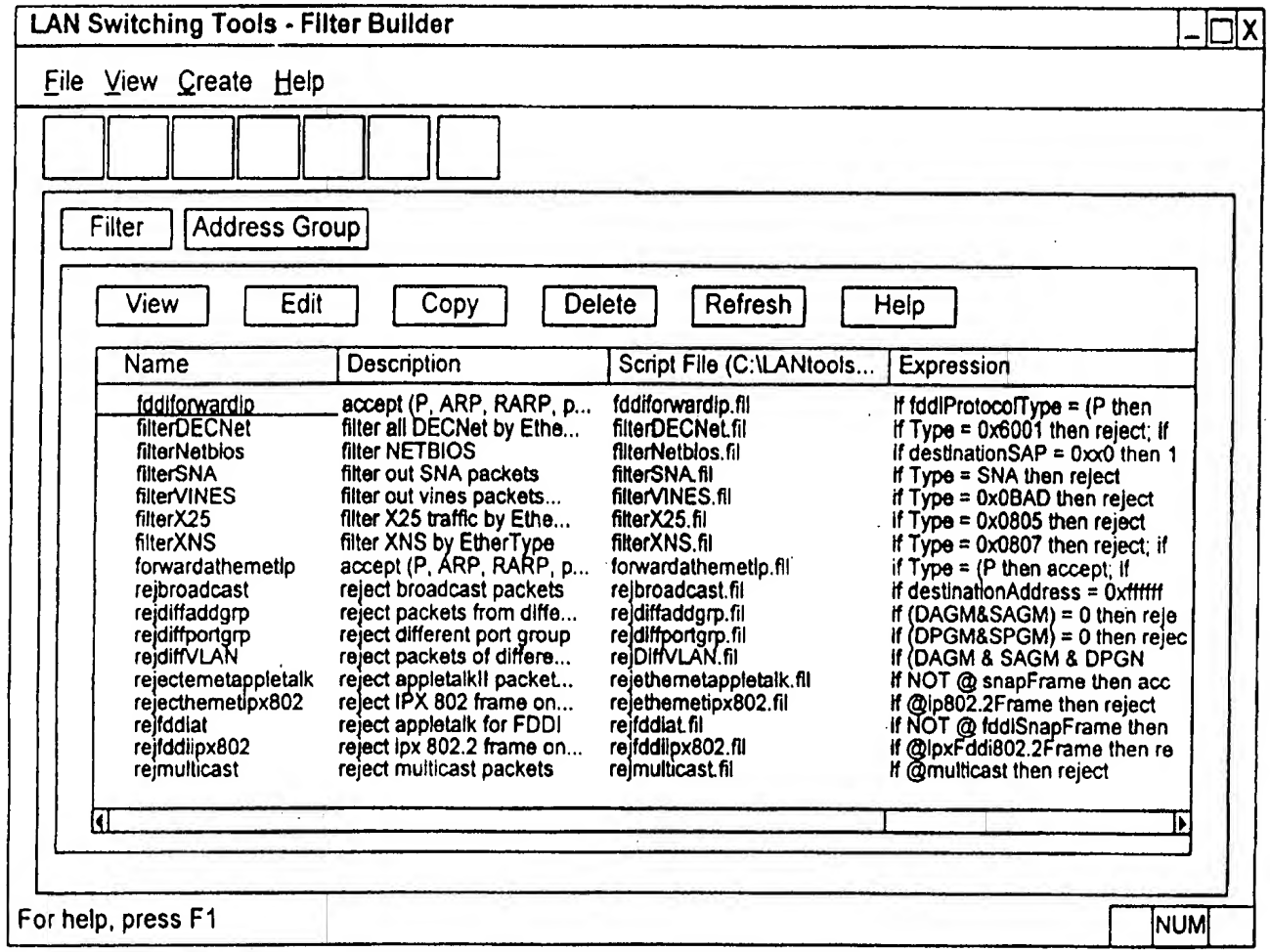

LAN Switching Tools - Filter Builder
File View Create Help
Filter
Address Group
View
Edit
Copy
Delete
Refresh
Help
Name
Description
Script File (C:\LANtools...
Expression
fddiforwardip
accept (P, ARP, RARP, p...
fddiforwardip.fil
If fddiProtocolType = (P then
filterDECNet
filter all DECNet by Ethe...
filterDECNet.fil
If Type = 0x6001 then reject; If
filterNetbios
filter NETBIOS
filterNetbios.fil
If destinationSAP = 0xx0 then 1
filterSNA
filter out SNA packets
filterSNA.fil
If Type = SNA then reject
filterVINES
filter out vines packets...
filterVINES.fil
If Type = 0x0BAD then reject
filterX25
filter X25 traffic by Ethe...
filterX25.fil
If Type = 0x0805 then reject
filterXNS
filter XNS by EtherType
filterXNS.fil
If Type = 0x0807 then reject; If
forwardathemetIp
accept (P, ARP, RARP, p...
forwardathemetIp.fil
If Type = (P then accept; If
rejbroadcast
reject broadcast packets
rejbroadcast.fil
If destinationAddress = 0xffffff
rejdiffaddgrp
reject packets from diffe...
rejdiffaddgrp.fil
If (DAGM&SAGM) = 0 then reje
rejdiffportgrp
reject different port group
rejdiffportgrp.fil
If (DPGM&SPGM) = 0 then rejec
rejdiffVLAN
reject packets of differe...
rejDiffVLAN.fil
If (DAGM & SAGM & DPGN
rejectemetappletalk
reject appletalkII packet...
rejethemetappletalk.fil
If NOT @ snapFrame then acc
rejecthemetipx802
reject IPX 802 frame on...
rejethemetipx802.fil
If @Ip802.2Frame then reject
rejfddiat
reject appletalk for FDDI
rejfddiat.fil
If NOT @ fddiSnapFrame then
rejfddiipx802
reject ipx 802.2 frame on...
rejfddiipx802.fil
If @IpxFddi802.2Frame then re
rejmulticast
reject multicast packets
rejmulticast.fil
If @multicast then reject
For help, press F1
NUM

FIG. 3

400

| Field | Description | Offset | Length (bytes) |
|---|---|---|---|
| destinationAddress | destination MAC address | 0 | 6 |
| sourceAddress | source MAC address | 6 | 6 |
| destinationOUI | Destination OUI | 0 | 3 |
| sourceOUI | Source OUI | 6 | 3 |
| type | type field | 12 | 2 |
| length | 802.3 length field | 12 | 2 |
| destinationSAP | Destination SAP address | 14 | 1 |
| sourceSAP | Source SAP address | 15 | 1 |
| SAP | both ssap & dsap field | 14 | 2 |
| control | LLC control | 16 | 1 |
| organizationID | Organization ID | 17 | 3 |
| protocolType | Network protocol | 20 | 2 |
| ipxSnapSourceNetwork | IPX Source Network | 40 | 4 |
| ipxSnapSourceNode | IPX Source Node | 44 | 6 |
| ipxSnapSourceSocket | IPX Source Socket | 50 | 2 |
| ipxFddi802Length | IPX Length (for FDDI 802.2 format) | 17 | 2 |
| ipxFddi802TransportControl | IPX Transport Control | 19 | 1 |
| ipxFddi802PacketType | IPX Packet Type | 20 | 1 |
| ipxFddi802DestNetwork | IPX Destination Network | 21 | 4 |
| ipxFddi802DestNode | IPX Destination Node | 25 | 6 |
| ipxFddi802DestSocket | IPX Destination Socket | 31 | 2 |
| ipxFddi802SourceNetwork | IPX Source Network | 33 | 4 |
| ipxFddi802SourceNode | IPX Source Node | 37 | 6 |
| ipxFddi802SourceSocket | IPX Source Socket | 43 | 2 |
| ipxFddiSnapLength | IPX Length (for FDDI SNAP format) | 22 | 2 |
| ipxFddiSnapTransportControl | IPX Transport Control | 24 | 1 |
| ipxFddiSnapPacketType | IPX Packet Type | 25 | 1 |
| ipxFddiSnapDestNetwork | IPX Destination Network | 26 | 4 |
| ipxFddiSnapDestNode | IPX Destination Node | 30 | 6 |
| ipxFddiSnapDestSocket | IPX Destination Socket | 36 | 2 |
| ipxFddiSnapSourceNetwork | IPX Source Network | 38 | 4 |
| ipxFddiSnapSourceNode | IPX Source Node | 42 | 6 |
| ipxFddiSnapSourceSocket | IPX Source Socket | 48 | 2 |
| appletalkDestNetworkNo | Appletalk Destination Network No (802.3 SNAP) | 26 | 2 |
| appletalkSourceNetworkNo | Appletalk Source Network No | 28 | 2 |
| appletalkDestNodeID | Appletalk Destination Node ID | 30 | 1 |
| appletalkSourceNodeID | Appletalk Source Node ID | 31 | 1 |
| appletalkDestSocketNo | appletalk Destination Socket no | 32 | 1 |
| appletalkSourceSocketNo | appletalk Source Socket no | 33 | 1 |
| appletalkDDPType | appletalk DDP Type | 34 | 1 |
| appletalkFddiDestNetworkNo | Appletalk Destination Network No (for FDDI) | 24 | 2 |
| appletalkFddiSourceNetworkNo | Appletalk Source Network No | 26 | 2 |
| appletalkFddiDestNodeID | Appletalk Destination Node ID | 28 | 1 |
| appletalkFddiSourceNodeID | Appletalk Source Node ID | 29 | 1 |
| appletalkFddiDestSocketNo | appletalk Destination Socket no | 30 | 1 |
| appletalkFddiSourceSocketNo | appletalk Source Socket no | 31 | 1 |
| appletalkFddiDDPType | appletalk DDP Type | 32 | 1 |

FIG. 4A

400 (continued)

| Field | Description | Offset | Length (bytes) |
|---|---|---|---|
| SNAP | (org ID+netprot) | 17 | 5 |
| fddiDestinationSAP | FDDI destination SAP | 12 | 1 |
| fddiSourceSAP | FDDI source SAP | 13 | 1 |
| fddiControl | FDDI LLC control | 14 | 1 |
| fddiOrganizationID | FDDI organization ID | 15 | 3 |
| fddiProtocolType | FDDI Network Protocol | 18 | 2 |
| fddiSNAP | FDDI SNAP | 15 | 5 |
| ipEthernetServiceType | IP ServiceType (for Ethernet) | 15 | 1 |
| ipEthernetTotalLength | IP Total Length (for Ethernet) | 16 | 2 |
| ipEthernetProtocolID | IP Protocol ID(for Ethernet) | 23 | 1 |
| ipEthernetSourceIPAddress | IP Source Address (for Ethernet) | 26 | 4 |
| ipEthernetDestinationIPAddress | IP Destination Address (for Ethernet) | 30 | 4 |
| ipFddiServiceType | IP ServiceType (for FDDI) | 21 | 1 |
| ipFddiTotalLength | IP Total Length (for FDDI) | 22 | 2 |
| ipFddiProtocolID | IP Protocol ID(for FDDI) | 29 | 1 |
| ipFddiSourceIPAddress | IP Source Address (for FDDI) | 32 | 4 |
| ipFddiDestinationIPAddress | IP Destination Address (for FDDI) | 36 | 4 |
| ipxRawLength | ipx Length (for 802.3 Raw format) | 16 | 2 |
| ipxRawTransportControl | IPX Transport Control | 18 | 1 |
| ipxRawPacketType | IPX Packet Type | 19 | 1 |
| ipxRawDestNetwork | IPX Destination Network | 20 | 4 |
| ipxRawDestNode | IPX Destination Node | 24 | 6 |
| ipxRawDestSocket | IPX Destination Socket | 30 | 2 |
| ipxRawSourceNetwork | IPX Source Network | 32 | 4 |
| ipxRawSourceNode | IPX Source Node | 36 | 6 |
| ipxRawSourceSocket | IPX Source Socket | 42 | 2 |
| IPXEthernetLength | IPX Length (for Ethernet II format) | 16 | 2 |
| IPXEthernetTransportControl | IPX Transport Control | 18 | 1 |
| IPXEthernetPacketType | IPX Packet Type | 19 | 1 |
| IPXEthernetDestNetwork | IPX Destination Network | 20 | 4 |
| IPXEthernetDestNode | IPX Destination Node | 24 | 6 |
| IPXEthernetDestSocket | IPX Destination Socket | 30 | 2 |
| IPXEthernetSourceNetwork | IPX Source Network | 32 | 4 |
| IPXEthernetSourceNode | IPX Source Node | 36 | 6 |
| IPXEthernetSourceSocket | IPX Source Socket | 42 | 2 |
| IPX802Length | IPX Length (for 802.2 format) | 19 | 2 |
| IPX802TransportControl | IPX Transport Control | 21 | 1 |
| IPX802PacketType | IPX Packet Type | 22 | 1 |
| IPX802DestNetwork | IPX Destination Network | 23 | 4 |
| IPX802DestNode | IPX Destination Node | 27 | 6 |
| IPX802DestSocket | IPX Destination Socket | 33 | 2 |
| IPX802SourceNetwork | IPX Source Network | 35 | 4 |
| IPX802SourceNode | IPX Source Node | 39 | 6 |
| IPX802SourceSocket | IPX Source Socket | 45 | 2 |
| IPXSnapLength | IPX Length (for 802.3 SNAP format) | 24 | 2 |
| IPXSnapTransportControl | IPX Transport Control | 26 | 1 |
| IPXSnapPacketType | IPX Packet Type | 27 | 1 |
| IPXSnapDestNetwork | IPX Destination Network | 28 | 4 |
| IPXSnapDestNode | IPX Destination Node | 32 | 6 |
| IPXSnapDestSocket | IPX Destination Socket | 38 | 2 |

FIG. 4B

500

| Field | Description |
|---|---|
| field.b:<offset> | 1 byte field at <offset) |
| field.w:<offset> | 2 byte field at <offset) |
| field.1:<offset> | 4 byte field at <offset) |
| field.a:<offset> | 6 byte field at <offset) |
| SAGM | Source Address Group Mask |
| DAGM | Destination Address Group Mask |
| SPGM | Source Port Group Mask |
| DPGM | Destination Port Group Mask |

FIG. 5

600

| Constant | Expression |
|---|---|
| IP | 0x0800 |
| AppletalkDDP | 0x809b |
| ARP | 0x0806 |
| RARP | 0x8035 |
| decMOP | 0x6002 |
| decLAT | 0x6004 |
| XNS | 0x0600 |
| IPX | 0x8137 |
| SNA | 0x80d5 |
| | |
| TCP | 0x06 |
| UDP | 0x11 |
| ICMP | 0x01 |
| EGP | 0x08 |
| OSPF | 89 |

FIG. 6

700

| Operator | Meaning | Precedence | associativity |
|---|---|---|---|
| NOT | logical not | highest | right to left |
| << | Left shift | | left to right |
| >> | right shift | | |
| <= | less than or equal to | | left to right |
| < | less than | | |
| >= | greater than or equal to | | |
| > | greater than | | |
| = | equal | | left to right |
| != | not equal | | |
| & | bit-wise AND | | left to right |
| ^ | bit-wise exclusive OR | | left to right |
| \| | bit-wise OR | | left to right |
| AND | x logical AND y | | left to right |
| OR | x logical OR y | Lower | left to right |

FIG. 7

802

- Ethernet II Framing on Ethernet/ATM LANE:

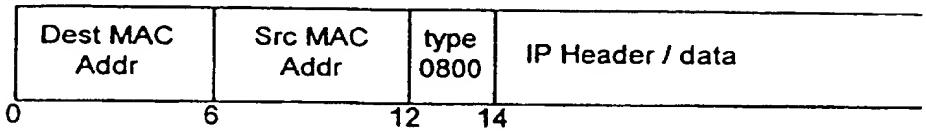



FIG. 8A

804

- FDDI SNAP Framing:

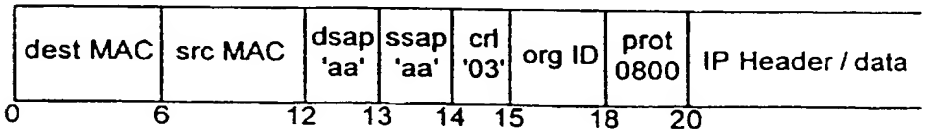



FIG. 8B

816

- Novell "RAW" 802.3 Framing:

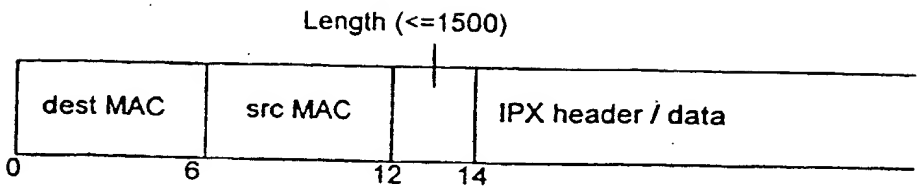



FIG. 8C

808

- Ethernet II Framing:

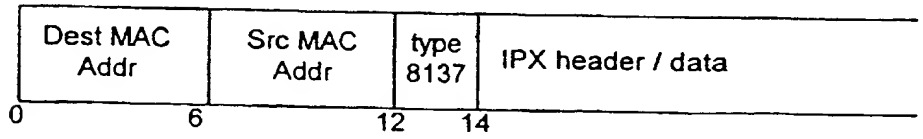



FIG. 8D

810

- LLC 802.2 Framing:

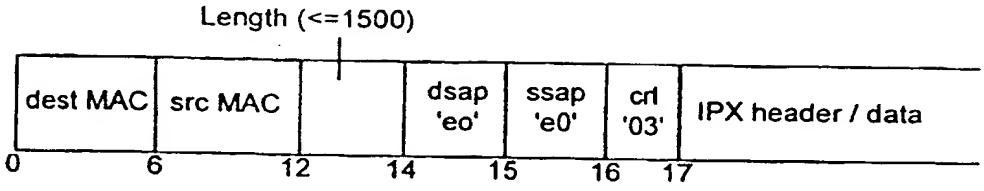



FIG. 8E

812

- 802.3 SNAP Framing:

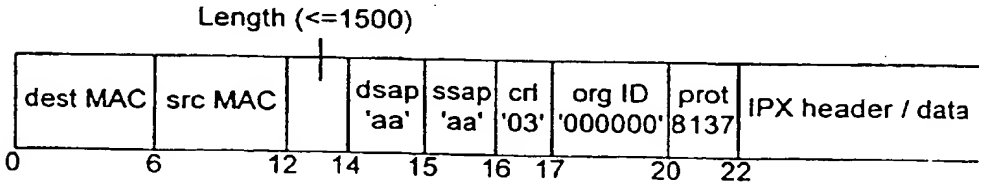



FIG. 8F

- FDDI 802.2 Framing:

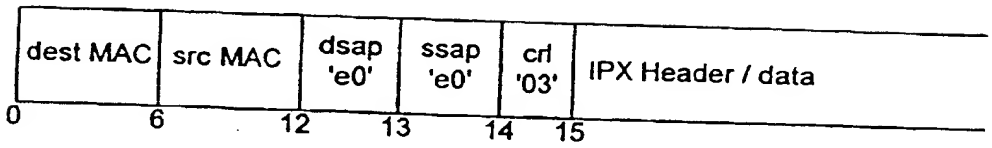
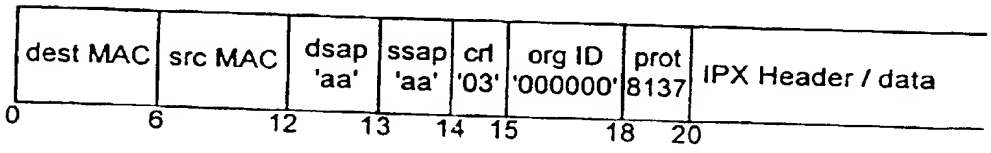



FIG. 8G

- FDDI SNAP Framing:




(iii) Appletalk packet

FIG. 8H

818

- 802.3 SNAP Framing:

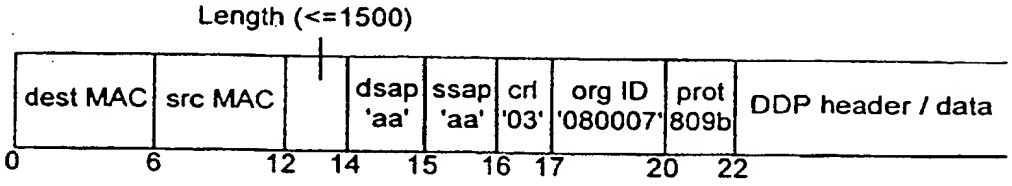



FIG. 8I

820

- FDDI SNAP Framing:

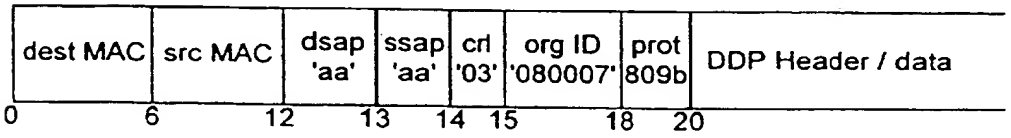



FIG. 8J

1100

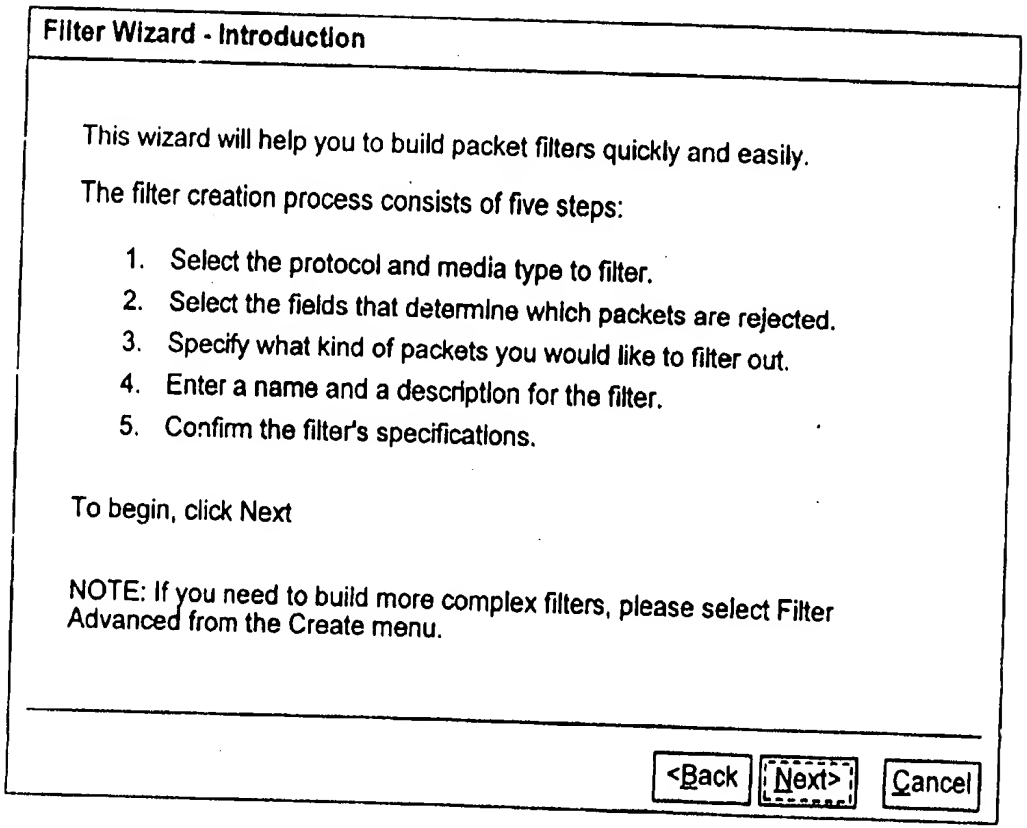

Filter Wizard - Introduction
This wizard will help you to build packet filters quickly and easily.
The filter creation process consists of five steps:
1. Select the protocol and media type to filter.
2. Select the fields that determine which packets are rejected.
3. Specify what kind of packets you would like to filter out.
4. Enter a name and a description for the filter.
5. Confirm the filter's specifications.
To begin, click Next
NOTE: If you need to build more complex filters, please select Filter Advanced from the Create menu.
<Back
Next>
Cancel

FIG. 9

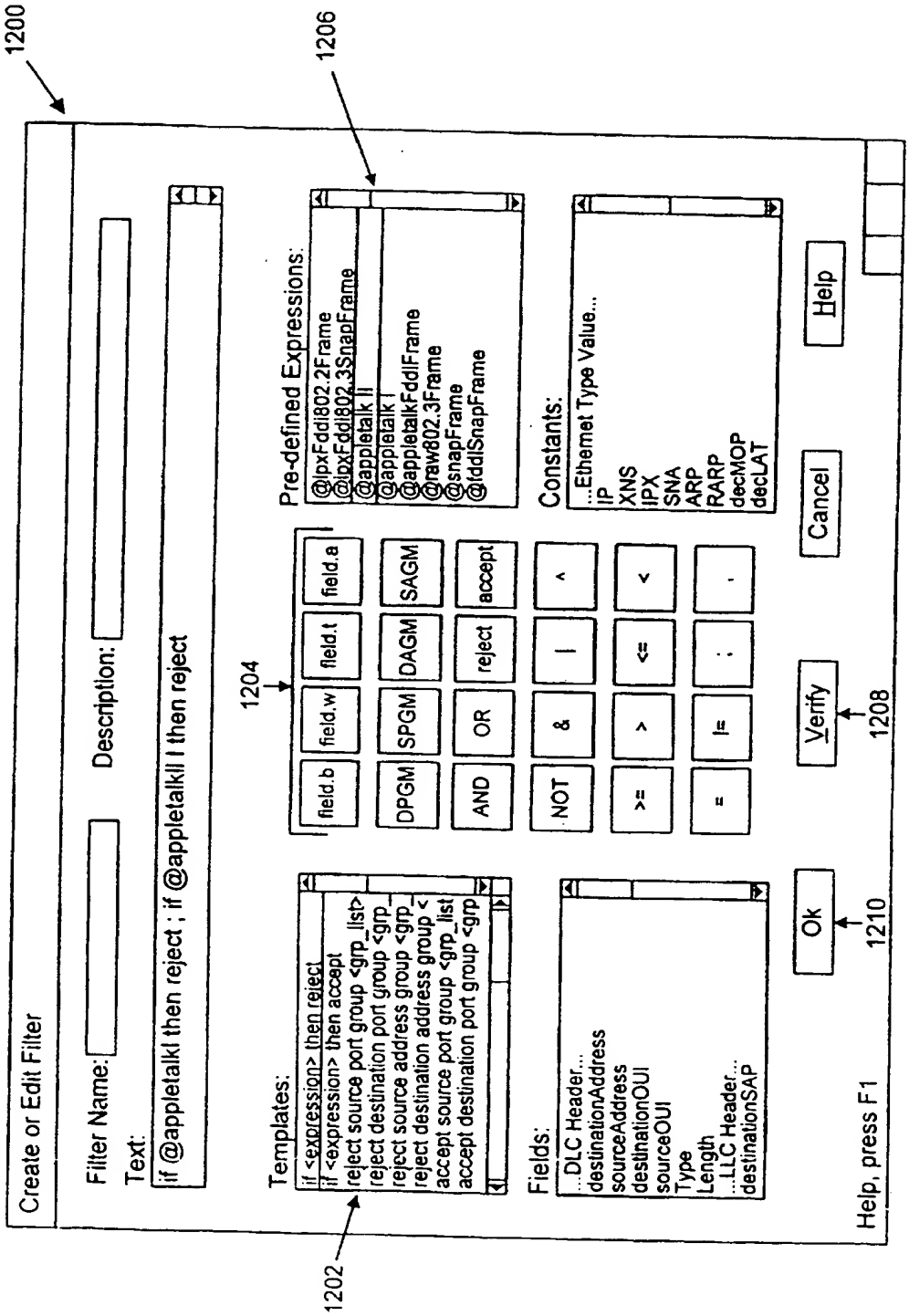

1200
1206
Create or Edit Filter
Filter Name:
Description:
Text:
if @appletalkI then reject ; if @appletalkII then reject
1204
Templates:
if <expression> then reject
if <expression> then accept
reject source port group <grp_list>
reject destination port group <grp_
reject source address group <grp_
reject destination address group <
accept source port group <grp_list
accept destination port group <grp
Pre-defined Expressions:
@ipxFddi802.2Frame
@ipxFddi802.3SnapFrame
@appletalk II
@appletalk I
@appletalkFddiFrame
@raw802.3Frame
@snapFrame
@fddiSnapFrame
field.b
field.w
field.t
field.a
DPGM
SPGM
DAGM
SAGM
AND
OR
reject
accept
NOT
&
|
^
>=
>
<=
<
=
!=
;
,
Constants:
...Ethernet Type Value...
IP
XNS
IPX
SNA
ARP
RARP
decMOP
decLAT
Fields:
...DLC Header...
destinationAddress
sourceAddress
destinationOUI
sourceOUI
Type
Length
...LLC Header...
destinationSAP
Ok
Verify
Cancel
Help
1202
1210
1208
Help, press F1

FIG. 10

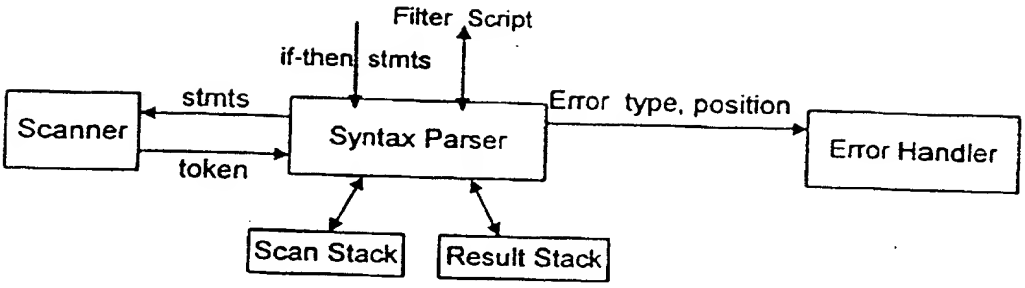

Filter Script
if-then stmts
stmts
Scanner
token
Syntax Parser
Error type, position
Error Handler
Scan Stack
Result Stack

FIG. 11

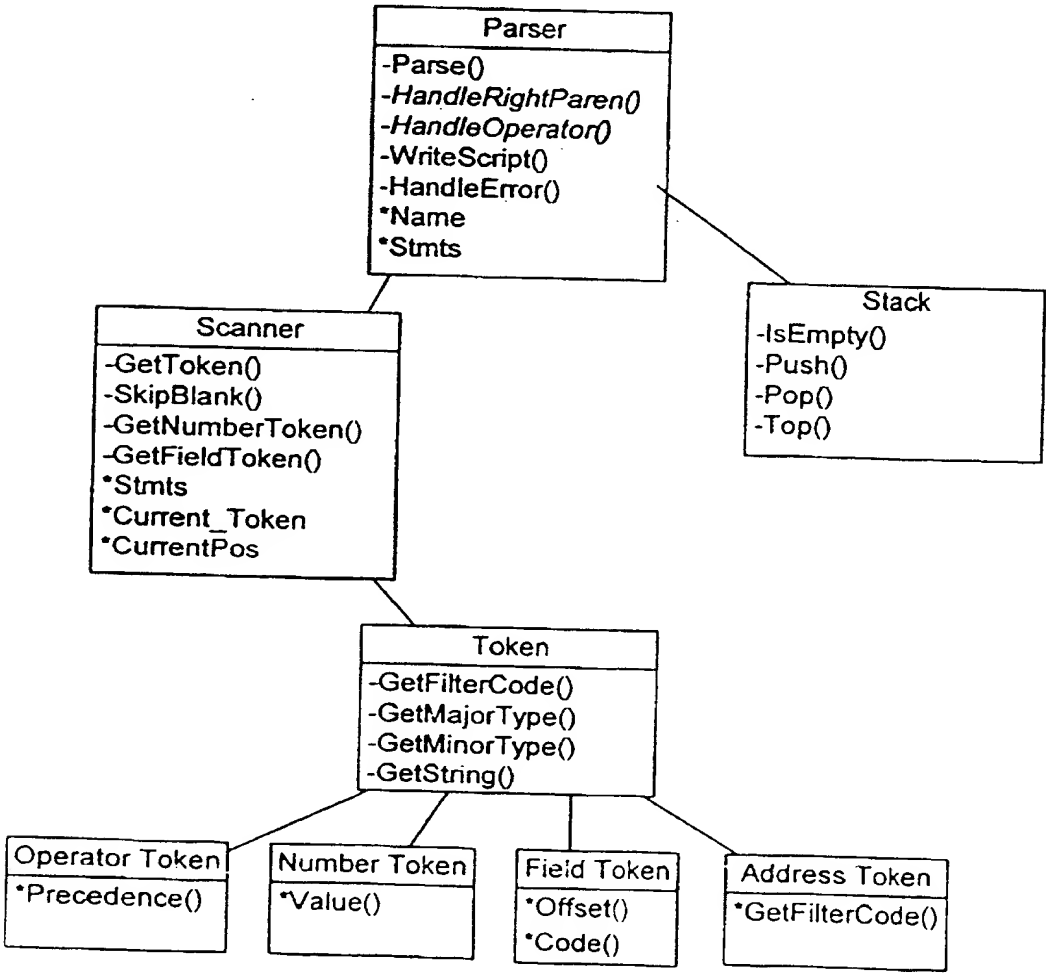

Parser
-Parse()
-HandleRightParen()
-HandleOperator()
-WriteScript()
-HandleError()
*Name
*Stmts
Scanner
-GetToken()
-SkipBlank()
-GetNumberToken()
-GetFieldToken()
*Stmts
*Current_Token
*CurrentPos
Stack
-IsEmpty()
-Push()
-Pop()
-Top()
Token
-GetFilterCode()
-GetMajorType()
-GetMinorType()
-GetString()
Operator Token
*Precedence()
Number Token
*Value()
Field Token
*Offset()
*Code()
Address Token
*GetFilterCode()

FIG. 12

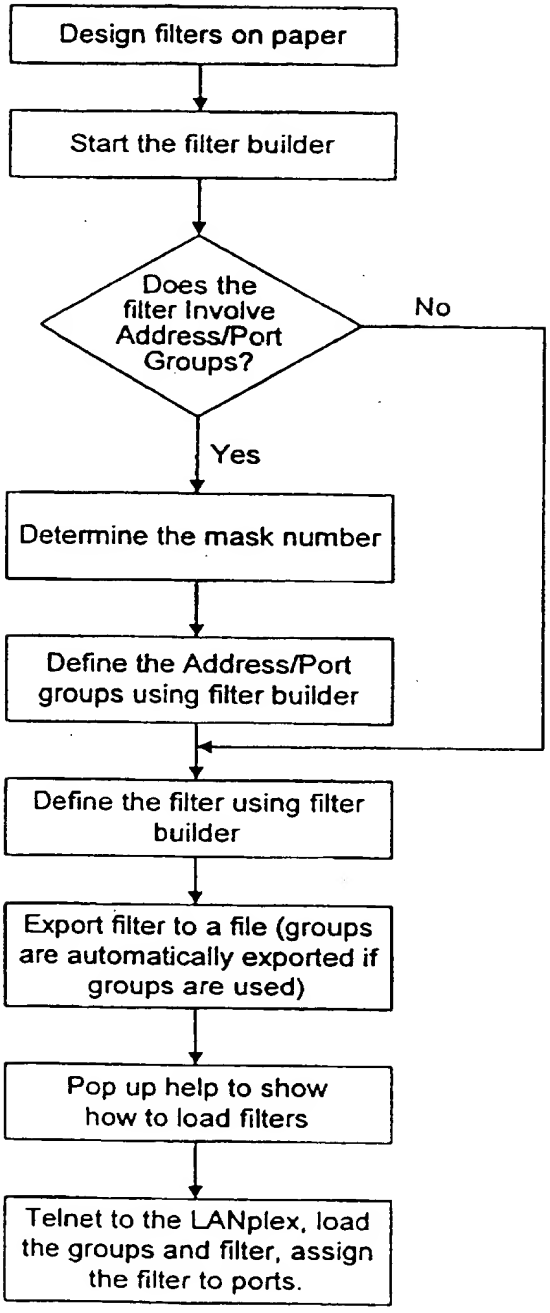

Design filters on paper
Start the filter builder
Does the filter Involve Address/Port Groups?
No
Yes
Determine the mask number
Define the Address/Port groups using filter builder
Define the filter using filter builder
Export filter to a file (groups are automatically exported if groups are used)
Pop up help to show how to load filters
Telnet to the LANplex, load the groups and filter, assign the filter to ports.

FIG. 13A

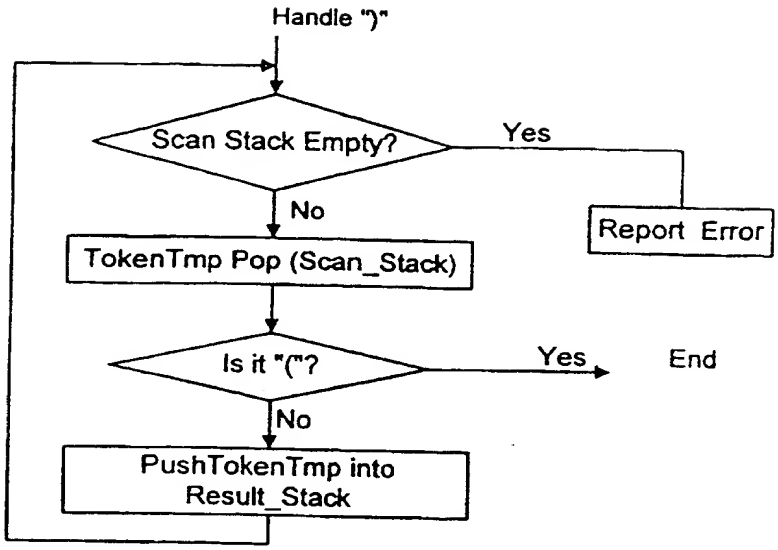
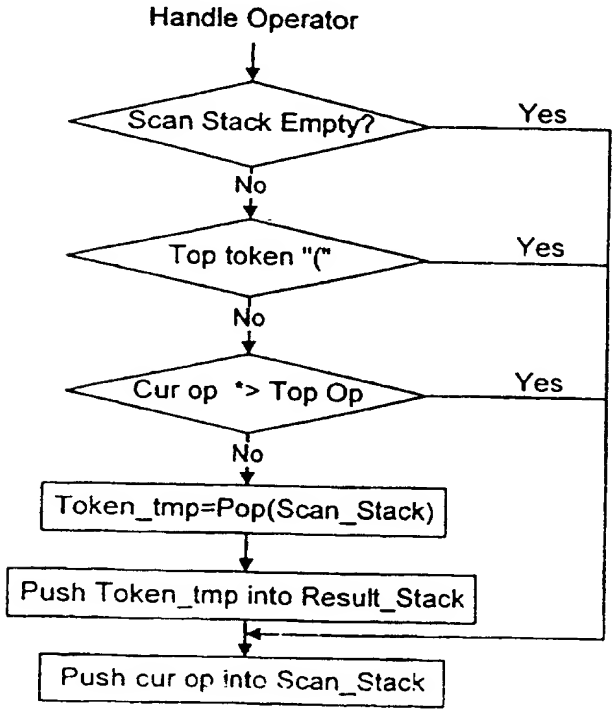



FIG. 13B




*> means operator 1 has precedence over operator 2.

FIG. 13C

```
statement ()
{
        if (lookahead == IF)
        {
                b-exp();
                if (lookahead == THEN)
                {
                        if (lookahead == REJECT) I (lookahead == ACCEPT)
                                finish;
                        else
                                Error("REJECT or ACCEPT is expected");
                }
                else
                        Error("THEN is expected");
        } else if (lookahead == REJECT) I (lookahead == ACCEPT)
        {
                action();
        }
        else
        {
                b-exp();
        }
}
```

FIG. 14

```
<filter>            ::=<statement> {;<statement>}
<statement>         ::=<b-exp> | IF <b-exp> THEN [REJECT|ACCEPT] | <action>
<b-exp>*            ::=<logical-and-exp> | <b-exp> OR <b-exp>
<logical-and-exp>*  ::=<inclusive-or-exp> | <logical-and-exp> AND <inclusive-or-exp>
<inclusive-or-exp>  ::=<exclusive-or-exp>
                        | <inclusive-or-exp> "|" exclusive-or-exp>
<exclusive-or-exp>  ::=<and-exp> | <exclusive-or-exp> ^ <and-exp>
<and-exp>           ::=<equality-exp> | <and-exp> & <equality-exp>
<equality-exp>      ::=<rel-exp> | <equality-exp> [ = |! = ] <rel-exp>
<rel-exp>           ::=<shift-exp> | <rel-exp> [ >= | > | <= | < ] <shift-exp>
<shift-exp>         ::=<not-exp> | <shift-exp> [ << | >> ] <not-exp>
<not-exp>           ::=not <operand>
<operand>           ::=(<inclusive-or-exp>) | <field> | <constant>
<action>            ::=[REJECT|ACCEPT] [SOURCE|DESTINATION]
                        [ADDRESS|PORT] GROUP <number-list>
<number-list>       ::=<number> {, <number>}
```

The above are defined under the assumption that logical-AND, logical-OR can be substituted by bit-wise-AND, bit-wise-OR respectively. Otherwise, the following definition will replace the two definitions with *:

```
<b-exp>             ::=<log-exp-list> | (<log-exp-list>)
<log-exp-list>      ::=<log-and-exp-list> | <log-or-exp-list>
<log-and-exp-list>  ::=<inclusive-or-exp> { AND <inclusive-or-exp>}
<log-or-exp-list>   ::=<inclusive-or-exp> { OR <inclusive-or-exp>}
```

FIG. 15

# A METHOD OF BUILDING A PACKET FILTER

## TECHNICAL FIELD

The present invention relates to the field of network management software. More specifically, the present invention relates to packet filters utilized in network systems.

## BACKGROUND ART

User-defined Packet Filters are a very powerful feature of networking switches. Unfortunately, conventional packet filter building methods are often difficult to perform and implement, and also error prone. For example, conventional packet building methods require the network manager or other individual building the packet filters to be intimately familiar with the basic elements(e.g. fields, operators, constands, and the like) used in packet filters. Such packet filter building knowledge is extensive. In addition, conventional packet filter building methods require the network manager or packet filter builder to have low-level programming language knowledge and skills because packet filters have to be constructed in complex design languages, such as Reverse Polish stack-oriented language. Thus the average network manager can not effectively build user-defined Packet Filters using conventional packet filter building process.

User-defined Packet Filters can be used to improve network performance and increase network security. This enables network managers to maximize return of investment of very expensive switches. However, due to the aforementioned complexity associated with conventional packet filter building, many network managers lack the ability and programming expertise to construct desired packet filters. Thus, it is wasteful to let such a powerful feature like user-defined Packet Filters not be fully utilized, or worse compromising network performance and security.

Accordingly, what is needed is a system and method for building User-Defined Packet Filters that is easy and intuitive to use and maintain, not solely constructed using complex design language, does not require extensive networking knowledge and can be used effectively by average network managers.

## DISCLOSURE OF THE INVENTION

A method and computer system are described herein for packet filter building wherein the packet filter not constructed solely using complex design languages. The present invention further provides a packet filter building system and method which is not error-prone. Furthermore, the present invention provides a packet filter building system and method which does not require a highly trained programmer for the implementation thereof, and a packet filter building system and method which can be effectively and efficiently utilized by a typical network manager.

Specifically, two embodiments of the invention are presented to assist users in the construction of a packet filter such that the user is able to build a packet filter without being extensively trained in all of the numerous parameters involved in packet filter construction.

In one embodiment, a wizard-type interface guides the users through the formation of a packet filter on a step-by-step basis. That is, in such an embodiment, the present invention displays tutorial information to the user and prompts the user for various information or tells the user how to proceed. Then a packet filter is automatically generated at the end. Thus, by employing such a wizard-type approach, the present embodiment is able to assist even the most novice network manager or other packet filter builder in the creation of desired packet filters.

In another embodiment, the present invention provides a "calculator-type" interface for the formation of a packet filter by a more advanced network manager or other packet filter builder. In this embodiment, the user can enter

expression-based statements by either typing them or by clicking/double-clicking desired fields, operators, constants, or pre-built filters. The expression-based statements indicate the characteristics of the packet filter which the user wishes to build. The present embodiment then converts the expression-based statements to the final packet filter in traditional filter builder language. Thus, even in this more advanced embodiment, the present invention allows a network manager or other packet filter builder to construct a packet filter without being extensively trained in or cognizant of all of the various parameters in packet filter construction.

These and other advantages of the present invention will no doubt become obvious to those of ordinary skill in the art after having read the following detailed description of the preferred embodiments which are illustrated in the various drawing figures.

## BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and form a part of this specification, illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention:

FIGURE 1 is a schematic diagram of an exemplary computer system used to perform steps of the packet filter builder (PFB) method in accordance with one embodiment of the present invention.

FIGURE 2 is a flow chart of steps performed in accordance with one embodiment of the present claimed PFB invention.

FIGURE 3 is an illustration of one embodiment of a graphical user interface provided in accordance with one embodiment of the present claimed PFB invention.

FIGURES 4-7 are tables of information used during the building of a packet filter including the basic elements (e.g. fields, operators, constants, and the like) used in the building of packet filters.

FIGURES 8A-8J are frames of information used during the building of a packet filter.

FIGURE 9 is an illustration of one embodiment of a wizard-type graphical user interface provided in accordance with one embodiment of the present claimed PFB invention for less-advanced packet filter builders.

FIGURE 10 is an illustration of one embodiment of a calculator-type graphical user interface provided in accordance with one embodiment of the present claimed PFB invention for more advanced packet filter builders.

FIGURE 11 is a schematic diagram of the architecture of the implementation of Filter Parsing and Conversion utilized in accordance with the present claimed invention.

FIGURE 12 is a schematic diagram of the main classes and methods used in Filter Parsing employed in accordance with the present claimed invention.

FIGURES 13A-13C are flowcharts of the Operator Precedence Parsing Algorithm used in the present claimed invention.

FIGURE 14 is an example of pseudocode demonstrating Predictive Parsing Method utilized in accordance with the present claimed invention.

FIGURE 15 is the Expression-based Language Syntax Definition in BNF utilized in accordance with the present claimed invention.

The drawings referred to in this description should be understood as not being drawn to scale except if specifically noted.

## BEST MODE FOR CARRYING OUT THE INVENTION

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. While the invention will be described in conjunction with the preferred embodiments, it will be understood that they are not intended to limit the invention to these embodiments. On the contrary, the invention is intended to cover alternatives, modifications and equivalents, which may be included within the spirit and scope of the invention as defined by the appended claims. Furthermore, in the following detailed description of the present invention, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be obvious to one of ordinary skill in the art that the present invention may be practiced without these specific details. In other instances, well known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

Some portions of the detailed descriptions which follow are presented in terms of procedures, logic blocks, processing, and other symbolic representations of operations on data bits within a computer memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. In the present application, a procedure, logic block, process, etc., is conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system. It has proved convenient at times, principally

for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "determining", "assisting", "loading", "storing" or the like, refer to the actions and processes of a computer system, or similar electronic computing device. The computer system or similar electronic computing device manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission, or display devices. The present invention is also well suited to the use of other computer systems such as, for example, optical and mechanical computers.

## COMPUTER SYSTEM ENVIRONMENT OF THE PRESENT PACKET FILTER BUILDER INVENTION

With reference now to Figure 1, portions of the present automatic packet filter builder (PFB) method are comprised of computer-readable and computer-executable instructions which reside, for example, in computer-usable media of a computer system. Figure 1 illustrates an exemplary computer system 100 used to perform the PFB method in accordance with one embodiment of the present invention. It is appreciated that system 100 of Figure 1 is exemplary only and that the present invention can operate within a number of different computer

systems including general purpose networked computer systems, embedded computer systems, and stand alone computer systems specially adapted for packet filter building.

System 100 of Figure 1 includes an address/data bus 102 for communicating information, and a central processor unit 104 coupled to bus 102 for processing information and instructions. System 100 also includes data storage features such as a computer usable volatile memory 106, e.g. random access memory (RAM), coupled to bus 102 for storing information and instructions for central processor unit 104, computer usable non-volatile memory 108, e.g. read only memory (ROM), coupled to bus 102 for storing static information and instructions for the central processor unit 104, and a data storage unit 110 (e.g., a magnetic or optical disk and disk drive) coupled to bus 102 for storing information and instructions. A input output signal unit 112 (e.g. a modem) coupled to bus 102 is also included in system 100 of Figure 1. System 100 of the present invention also includes an optional alphanumeric input device 114 including alphanumeric and function keys is coupled to bus 102 for communicating information and command selections to central processor unit 104. System 100 also optionally includes a cursor control device 116 coupled to bus 102 for communicating user input information and command selections to central processor unit 104. System 100 of the present embodiment also includes an optional display device 118 coupled to bus 102 for displaying information.

Optional display device 118 of Figure 1, utilized with the present PFB method, may be a liquid crystal device, cathode ray tube, or other display device suitable for creating graphic images and alphanumeric characters recognizable to a user. Optional cursor control device 116 allows the computer user to

dynamically signal the two dimensional movement of a visible symbol (cursor) on a display screen of display device 118. Many implementations of cursor control device 116 are known in the art including a trackball, mouse, touch pad, joystick or special keys on alphanumeric input device 114 capable of signaling movement of a given direction or manner of displacement. Alternatively, it will be appreciated that a cursor can be directed and/or activated via input from alphanumeric input device 114 using special keys and key sequence commands. The present invention is also well suited to directing a cursor by other means such as, for example, voice commands. A more detailed discussion of the present PFB method is found below.

## GENERAL DESCRIPTION OF THE PRESENT PACKET FILTER BUILDER METHOD

With reference next to Figure 2, an flow chart 200 of exemplary steps used by the present PFB method is shown. Flow chart 200 includes processes of the present invention which, in one embodiment, are carried out by a processor under the control of computer-readable and computer-executable instructions. The computer-readable and computer-executable instructions reside, for example, in data storage features such as computer usable volatile memory 106 and/or computer usable non-volatile memory 108 of Figure 1. The computer-readable and computer-executable instructions are used to control, for example, the operation and functioning of central processing unit 104 of Figure 1. Although specific steps are disclosed in the flow chart of Figure 2, such steps are exemplary. That is, the present invention is well suited to performing various other steps or variations of the steps recited in Figure 2. The steps of Figure 2 will be described in conjunction with Figures 3-15.

With reference again to Figure 2, in step 202, a user of the present invention determines the initial design of the packet filter to be built. For example, the network manager or other designer of the packet filter writes down the features or functions of the desired packet filter. Although the present embodiment specifically recites writing down the features or functions of the packet filter to be built, it will be understood that the present invention is well suited to a user who does not first write down the features or functions of the packet filter to be built.

In step 204 of Figure 2, the user of the present invention initiates the present PFB invention. In the present embodiment, the user initiates the present PFB invention by, for example, using cursor control device 116 to select a PFB icon displayed on optional display device 118. It will be understood, however, that the present invention is well suited to using various other methods to initiate the present PFB invention. The present PFB invention provides a graphical user interface which guides the user through the creation of the desired packet filter. In one embodiment, designed for a more experienced or more highly trained network manager, the present invention provides a "calculator-type" interface. The calculator-type interface provided by the present PFB invention will be described and illustrated below in detail. In another embodiment, the present invention provides a "wizard-type" interface which guides the user through the formation of a packet filter on a step-by-step basis. The wizard-type interface provided in one embodiment the present PFB invention will be described and illustrated below in detail.

Referring next to Figure 3, an illustration of the present embodiment of a graphical user interface 300 provided by the present PFB invention is shown.

Thus, in step 204 of the present invention, when the user initiates the present PFB, GUI 300 appears, for example, on optional display device 118 of Figure 1. The present PFB invention utilizes a user-friendly, expression-based language to define packet filters. More specifically, each packet filter can be defined in a series of if-then statements. In each statement an expression is tested, then a specified action (i.e. a "reject" or an "accept") will take place. For example, if the user wants to define a filter to discard all Appletalk packets (Phase I and Phase II) using the expression-based language, such a packet filter will be defined by the present PFB invention in the following manner:

```
if (appletalkI or appleTalkII) then reject
or
if(appletalkI) then reject;
if(appletalkII) then reject.
```

The present PFB invention translates the user-friendly if then statement into the complex filter building language such that the user's expression-based if/then statements are converted into the requisite complex filter building language. As an example, the present PFB invention translates the above listed if/then statements into the following filter:

```
pushField.w          12
pushLiteral.w        0x809b
eq
reject
pushField.w          12
pushLiteral.w        1500
gt
accept
pushField.a          16
pushLiteral.a        0x03080007809b
ne
```

With conventional filter building methods, the above script has to be typed exactly as it is. As you can see, it requires extensive networking and programming knowledge and skill. And it is easy to make mistakes. However, in the present PFB invention, such filter script is invisible to the user. In so doing, the present

invention allows a user to build a desired packet filter using only intuitive if/then statements. Numerous, additional examples of such packet filters constructed using user-submitted if/then statements are given below in conjunction with step 212.

With reference next to Figures 4-7, tables 400-700 of information used during the building of a packet filter are shown. More specifically, tables 400-700 recite the basic elements (e.g. fields, operators, constants, and the like) used in the building of packet filters. In conventional packet filter building, the network manager, or person building the packet filter, would have to be intimately familiar with the information displayed in each of tables 400-700 of Figures 4-7, respectively. However, the present PFB invention eliminates the need for the network manager or other packet filter builder to be well-versed in such information in order to design and build packet filters. That is, the present PFB invention lowers the "network knowledge threshold" required to effectively design and build packet filters. In so doing, the present PFB invention enables the average network manager to build packet filters with the expertise and efficiency of a highly trained network programmer.

Additionally, with reference now to Figures 8A-8J, frames 802-820 of information used during the building of a packet filter are shown. As mentioned above in connection with Figures 4-7, in conventional packet filter building, the network manager, or person building the packet filter, would have to be intimately familiar with the information displayed in each of frames 802-820 of Figures 8A-8J, respectively. Thus, as mentioned above, the present PFB invention lowers the "network knowledge threshold" required to effectively design and build packet filters. In so doing, the present PFB invention enables the average network

manager to build packet filters with the expertise and efficiency of a highly trained network programmer.

Referring again to Figure 3, the GUI of the present PFB invention allows the user to readily view Filters, Port Groups, and Address Groups, or use the menu to create new Filters, Port Groups, and/or Address Groups. As shown in GUI 300 of Figure 3, a user of the present PFB invention is able to readily access packet filter building information, and initiate the packet filter building process. As an example, in the present embodiment of GUI 300, the following information can be retrieved by selecting either the File, View, Create, or Help:

| File | View | Create | Help |
|---|---|---|---|
| Find Filters... | Tool bars | Filter by wizard... | Help Info. |
| Change Script Directory | Status Bar | Filter Advanced... | |
| Start Telnet Session... | | Address Group... | |
| | | Port Group | |

It will be understood, however, that the present PFB invention is well suited to providing various additional and/or other information in GUI 300.

With reference next to step 206, the user of the present invention must determine whether the packet filter to be built involves address/port groups. If the packet filter to be built does not involve such address/port groups, the present PFB invention proceeds to step 212. If the packet filter to be built does involve address/port groups, the present invention proceeds to step 208.

In step 208, the user of the present PFB invention must determine the necessary mask number(s). The mask number is the bit number in the address group mask with which the user would like to associate a particular group. Each group takes up one bit of the 32 bits provided for address group administration in

the present embodiment of the PFB invention. Although 32 bits are provided for address group administration in this embodiment, the present invention is well suited to providing a different number of bits such as, for example, 64 bits for address group administration. In this embodiment of the PFB invention, this number is limited to the mask bits which have not been assigned on the selected slots. Furthermore, in the present embodiment, if an address group is loaded on multiple slots, the same bit in the address group mask will be used on each of the slots. Also, in this embodiment of the PFB invention, MAC addresses can be entered in either canonical, FDDI format, or hexadecimal which will be converted and displayed in canonical format.

Referring now to step 210, the present PFB invention then requires the user to define the address and port groups.

Referring now to step 212, the user of the present PFB invention defines the packet filters to be built. As mentioned above, the present invention allows the user to define the packet filters to be built using expression-based if/then statements. The present PFB invention translates the user-friendly if then statements into a complex filter building language such that the user's expression-based if/then statements are converted into the requisite complex filter building language. However, in the present PFB invention, such conversion is invisible to and/or hidden from the user. The following exemplary list recites several frequently requested packet filter types, and illustrates how the user's expression-based if/then statements are converted into the requisite complex filter building language.

Predefined Packet Filters Provided by the Present PFB Invention

A. PHYSICAL PORT FILTERING

1. Different Port Groups(Reject):

To discard packages of different port group.

[User Enters]

if(SPGM & DPGM) = 0 then reject

[Variables]

reject/accept

[Packet Filter Language]

```
Name            "Filter different address group
pushSPGM
pushDPGM
and
pushLiteral.l   0
ne
```

2. Source/Destination Port Groups:

To reject packets from port group 3 and 8.

[User Enters]

reject source port group 3, 8

[Variables]

Source/Destination

accept/reject

group(s)

[Packet Filter Language]

```
pushSPGM
        pushLiteral.l0x0084
        and
        pushLiteral.l0
        eq
```

B. MAC LAYER FILTERING

1. Different Address Groups(Reject):

To discard packages of different address group.

[User Enters]

if (SAGM & DAGM) =0 then reject

[Variables]

accept/reject

```
[Packet Filter Language]
Name                "Filter different port group"
pushSAGM
pushDAGM
and
pushLiteral.l       0
ne
```

2. Source/Destination Address Groups:
To reject packets from address group 3 and 8.

```
[User Enters]
reject source address group 3, 8

[Variables]
Source/Destination
reject/accept
address group(s)

[Packet Filter Language]
Name                "reject source address group 3,8"
pushSAGM
pushLiteral.l       0x0084
and
pushLiteral.l       0
eq
```

3. Source Address Filter:
This filter operates on the source address field of a frame. It rejects packets from station 00-DE-AD-00-00-02.

```
[User Enters]
if(saddr = 0x00DEAD000002) then reject

[Variables]
MAC address
reject/accept

[Packet Filter Language]
name                    "srcAddr_00DEAD000002_reject"
pushField.a         6                   # Get the source address
pushLiteral.a       0X00DEAD000002# Load desired address
ne                                      # Check for a match
```

4. Destination Address Filter:
This filter operates on the destination address field of a frame. It rejects packets to station 00-De-Ad-00-00-02.

```
[User Enters]
if(daddr = 0X00DEAD000002) then reject
```

```
[Variables]
MAC address
reject/accept

[Packet Filter Language]
name              "destAddr_00DEAD000002_reject"
pushField.a       0                          # Get the destination address
pushLiteral.a     0x00DEAD000002 # Load desired address
ne                                           # Check for a match
```

5. Source OUI:
This filter operates on the source network address field of a frame. It rejects packets from stations with an OUI of 00-DE-AD.

```
[User Enters]
if(SOUI = 00-DE-AD) then reject

[Variables]
SOUI
reject/accept

[Packet Filter Language]
name          "srcAddr_OUI=00DEAD_reject"
pushField.l       0                      # Get the first 4 bytes of the source address
pushLiteral.l 0xffffff00          # Setup mask to isolate first 3 bytes
and                                      # Top of stack now has OUI
pushLiteral.l 0x00DEAD00      # Load desired OUI value
ne                                       # Check for a match
```

6. Destination OUI:
This filter operates on the destination network address field of a frame. It rejects packets to be forwarded to stations with an OUI of 00-DE-AD.

```
[User Enters]
if(DOUI = 00-DE-AD) then reject

[Variables]
SOUI
reject/accept

[Packet Filter Language]
name "dstAddr_OUI=00DEAD_reject"
pushField.l       0                      # Get first 4 bytes of destination address
pushLiteral.l 0xffffff00          # Setup mask to isolate first 3 bytes
and                                      # Top of stack now has OUI
pushLiteral.l 0x00dead00      # Load desired OUI value
ne                                       # Check for a match
```

7. Multicast Filter:
This filter operates on the destination address field of a frame. It rejects all multicast packets.
[User Enters]
if(daddr and 0x01) = 0x01 then accept

[Variables]
Accept/reject
source/destination

[Packet Filter Language]
```
name "dstAddrMulticast_reject"
pushField.b     0           # Get the first byte of the destination address
pushLiteral.b   0x01        # Setup multicast mask
and                         # Isolate the multicast bit
pushLiteral.b       0x01        # Setup multicast bit
ne                          # Check for a multicast frame
```

8. Broadcast Filter:
This filter operates on the destination address field of a frame. It forwards all broadcast packets.
[User Enters]
if(daddr = 0xffffffffff) then accept

[Variables]
accept/reject
source/destination

[Packet Filter Language]
```
name                            "dstAddrBroadcast_forward"
pushField.a     0               # Get the destination address
pushLiteral.a   0xffffffffff    # Setup broadcast value
eq          # Check for a non-broadcast frame
```

9. Ethernet IP:
This filter operates on the type field of a frame. It allows packets to be forwarded that are IP frames. To customize this filter to another type value, change the literal value loaded in the pushLiteral.w instruction.
[User Enters]
if (type = ip) then accept

[Variables]
Accept/reject

[Packet Filter Language]
```
name            "EthernetIPaccept"
pushField.w     12          # Get the type field
pushLiteral.w   0x0800      # Load IP type value
```

```
eq                                  # Check for a match
```

10. "RAW" IEEE 802.3:

```
[User Enters]
if (length <=1500) then accept

[Variables]
accept/reject

[Packet Filter Language]
name            "raw_1EEE802.3"
pushField.w     12
pushLiteral.w   1500
ge
```

11. EthernetII IPX:

This filter operates on the type field of a frame. It allows packets to be forwarded that are IpX frames. To customize this filter to another type value, change the literal value loaded in the pushLiteral.w instruction.

```
[User Enters]
if (type = IPX) then reject

[Variables]
Accept/reject

[Packet Filter Language]
name            "EthernetII_IPX_reject"
pushField.w     12                  # Get the type field
pushLiteral.w   0x8137              # Load IPX type value
ne                                  # Check for a match
```

12. IEEE 802.2 IPX:

This filter rejects IPX 802.2 frames.

```
[User Enters]
ifIPX8O2.2 then reject or
if (sap = OxeOeO) and (ctl = 0x03) then reject

[Variable]
Accept/reject

[Packet Filter Language]
name                    "IEEE802.2_reject"
pushField.l             14              # Get the dsap, ssap, ctrl field
pushLiteral.l   0xffffff00      #
and                             #
pushLiteral.l           0xaaaa0300 # Load value
ne                              # Check for a match
```

13. IEEE 802.3:

This filter rejects IEEE 802.3 frames.
[User Enters]
if 802.3 then reject
or
if(sap = 0xaaaa) and (ctl = 0x03) then reject

[Variables]
Accept/reject

[Packet Filter Language]
```
name                "IEEE802.3_reject"
pushField.l         14                  # Get the dsap, ssap,ctrl field
pushLiteral.l 0xffffff00          #
and
pushLiteral.l 0xaaaa0300          #Load IEEE802.3 value
ne
```

14. IPX 802.3:
This filter filters IPX 802.3 SNAP frames.
[User Enters]
if 802.3 and netprot = IPX then reject

[Variables]
Accept/reject

[Packet Filter Language]
```
name                "IEEE802.3_reject"
pushField.l         14                  # Get the dsap, ssap,ctrl field
pushLiteral.l 0xffffff00          #
and
pushLiteral.l 0xaaaa0300 #Load IEEE802.3 value
ne
accept
pushField.w  20                         # get the protocol type
pushLiteral.w       0x8137              # Load IPX type
ne
```

15. Appletalk I filter:
This filter operates on the type field of a frame. It allows packets to be forwarded that are Appletalk frames. To customize this filter to another type value, change the literal value loaded in the pushLiteral.w instruction.
[User Enters]
if (type = appletalk) then reject

[Variable]
Accept/reject

```
[Packet Filter Language]
name                "AppletalkI_reject"
pushField.w         12                  # Get the type field
pushLiteral.w       0x809b              # Load Appletalk type value
ne                                      # Check for a match
```

16. Appletalk II (appletalk 802.3 snap) Filter:
This filter rejects appletalkII frames.

```
[User Enters]
if appletalkII then reject

[Variables]
Accept/reject

[Packet Filter Language]
name                "appletalkII"
pushField.w         14                  # Get the type field
pushLiteral.w       0xaaaa              # 802.3
eq                                      # Check for a match
accept
pushField.a         16
pushLiteral.a       0x03080007809b
ne
```

17. Maximum Length Filter:
This filter operates on the length field of a frame. It allows packets to be forwarded that are less than 400 bytes in length. To customize this filter to another length value, change the literal value loaded in the pushLiteral.w instruction.

```
[User Enters]
if(Length <== 400) then accept

[Variables]
length
accept/reject

[Packet Filter Language]
name "Forward <= 400"
pushField.w         12              # Get length field
pushField.w         400             # load length limit
le
```

18. Minimum Length Filter:
This filter operates on the length field of a frame. It allows packets to be forwarded that are greater than 900 bytes in length. To customize this filter to another length value, change the literal value loaded in the pushLiteral.w instruction.

```
[User Enters]
if(length >= 900) then accept
```

```
[Variables]
length
accept/reject

[Packet Filter Language]
name "Forward >= 900"
pushField.w        12          # Get length field
pushField.w        900         # load length limit
ge
```

19. FDDI 802.3:

This filter rejects FDDI IEEE 802.3 frames.

```
[User Enters]
ifFDDI_802.3 then reject or
if (FDD_sap = Oxaaaa) and (FDDI_ctl = 0x03) then reject

[Variables]
Accept/reject

[Packet Filter Language]
name               "FDDI_802.3_reject"
pushField.l        12                  # Get the dsap~ ssap,ctrl field
pushLiteral.l 0xffffff00       #
and
pushLiteral.l 0xaaaa0300 #Load IEEESO2.3 value
ne
```

20. FDDI IP:

This filter rejects FDDI IEEE 802.3 SNAP frames.

```
[User Enters]
ifFDDI_802.3 and netpro = IP then reject

[Variable]
Accept/reject

[Packet Filter Language]
name               "FDDI_IP_reject"
pushField.l        12                  # Get the dsap. ssap,ctrl field
pushLiteral.l 0xffffff00       #
and
pushLiteral.l OxaaaaO3OO       #Load IEEESO2.3 value
ne
accept
pushField.w        18
pushLiteral.w      0x0800
ne
```

21. FDDI IPX 802.2:
This filter rejects FDDI IPX 802.2 frames.
[User Enters]
if FDDI_IPX802.2 then reject
or
if (sap OxeOeO) and (ctl = 0x03) then reject

[Variables]
Accept/reject

[Packet Filter Language]
```
name                "FDDI_802.2_reject"
pushField.l         12                  # Get the dsap, ssap, ctrl field
pushLiteral.l 0xffffff00          #
and
pushLiteral.l OxeOeOO3OO          # Load value
ne                                      # Check for a match
```

22. FDDI IPx 802.3:
This filter rejects FDDI IPX 802.3 SNAPframes.
[User Enters]
if FDDI_802.3 and FDDI_netprot = IPX then reject

[Variables]
Accept/reject

[Packet Filter Language]
```
name                    "FDDT_802.3_IPX_reject"
pushField.l             12                  # Get the dsap, ssap, ctrl field
pushLiteral.l       0xffffff00          #
and                                         #
pushLiteral.1           OxaaaaO3OO          # Load value
ne
accept
pushField.w         18
pushLiteral.w           0x8137
ne
```

23. FDDI Appletalk:
This filter rejects FDDI appletalk frames.
[User Enters]
if FDDI_802.3 and FDDI_netprot = appletalk then reject

[Variables!
Accept/reject

[Packet Filter Language]
```
name                "FDDI_appletalk_reject"
pushField.l         12                      # Get the dsap, ssap, ctrl field
```

```
pushLiteral.l 0xffffff00                #
and                                             #
pushLiteral.l 0xaaaa0300                # Load value
ne
accept
pushField.w     18
pushLiteral.w   0x809b
ne
```

C. NETWORK LAYER FILTERING

1. TCP Filter:

This filter discards all TCP packets.

[User Enters]

if(netprot = 1? and tranprot = TCP) then reject

[Variables]

Accept/reject

[Packet Filter Language]

```
name                                    "ethIP_TcpOrUdp_reject.pfl"
pushField.w     12              # Get the type field
pushLiteral.w   0x0800          # Load 'P type value
ne
accept
pushField.b     23              # Get the protocol type in IF header
pushLiteral.b   0x06            # Load TCP protocol type
ne                              # Mismatch, accept
```

2. UDP Filter:

This filter discards all UDP packets.

[User Enters]

if(netprot = IP and tanprot = UDP) then reject

[Variables]

Accept/reject

[Packet Filter Language]

```
name                                    "ethIP_TcpOrUdp_reject.pfl"
pushField.w     12              # Get the type field
pushLiteral.w   0x0800                  # Load IP type value
ne
accept
pushField.b     23                      # Get the protocol type in I? header
pushLiteral.b   0x11                    # Load UDP protocol type
ne                                      # Mismatch, accept
```

3. Subnet Directed Broadcast Filter (reject):
This filter operates on the destination IP address of an IP frame. It discards IP (class B with 8 bit subnet) subnet broadcast packets (x.x.x.255).
[User Enters]
if (IF and field.b.33 = Oxif) then reject

[Variables]
Accept/reject

[Packet Filter Language]
```
name            "EthIP_subnetBcast_reject.pfl"
pushField.w     12          # Get the type field
pushLiteral.w   0x0800      # Load IP type value
ne
accept
pushField.b     33          # Get last byte of Dest IP address
pushLiteral.b   Oxff        # Load broadcast byte 255
ne              # Mismatch, then forward
```

4. Filter 6 bytes at byte 56 (forward):
This filter operates on the 56th byte of a frame. Filters within the first 20 bytes are handled differently than the rest of the packet.
[User Enters]
if field.a:56 = Ox00ccccccccc then accept

[Packet Filter Language]
```
name                "Filt6BytesAtByte56_forward"
pushField.a         56
pushLiteral.a       OxO0ccccccccc
eq
```

Hence, the present PFB invention allows the user to enter expression-based statements an construct complex packet filters without requiring that the user be well-versed in complex and error prone programming languages. Additionally, the present PFB invention does not require that the user be extensively trained in or cognizant of all of the various parameters in packet filter construction. Furthermore, although the above-cited examples explicitly recite that the user enter an expression-based statement, the present invention is also well suited to having the user select such statements through various other methods. For example, the present invention is also well suited to having the user select statements via pull-down windows, double-clicking on the desired filter type, and

the like. Although such packet filter types are specifically recited above, it will be understood that the above-listed packet filter types are exemplary, and that the present invention is well suited to having various other packet filter types. It will also be understood that the present PFB invention allows the user to build a plurality of packet filters if desired.

Referring next to step 214, the present PFB invention then prompts the user to load the stored packet filter. Although such an approach is employed in the present embodiment, the present invention is also well suited to automatically loading the constructed packet filter for the user. In such an embodiment, the particulars of the loading of the packet filter are defined, for example, by user entered information.

With reference next to Figure 9, another GUI 1100 is shown in accordance with the present claimed invention. In this embodiment, the present invention provides a "wizard-type" interface which guides the user through the formation of a packet filter on a step-by-step basis. In such an embodiment, the present PFB invention guides the user through the packet filter building method in a step-by-step process. That is, in such an embodiment, the present invention displays tutorial information to the user as shown in GUI 1100. Next, the present embodiment of the PFB invention prompts the user for various information or tells the user how to proceed. Thus, by employing such a wizard-type approach, the present PFB invention is able to assist even the most novice network manager or other packet filter builder in the creation of desired packet filters.

With reference next to Figure 10, another GUI 1200 is shown in accordance with the present claimed invention. In this embodiment, the present invention

provides a "calculator-type" interface for the formation of a packet filter by a more advanced network manager or other packet filter builder. In the embodiment of Figure 10, the user can select fields, operators, constants or pre-built filters by either double clicking on the item in the list box 1202 or operator buttons 1204. The corresponding text will appear in the editable box 1206, the user can then fill in variables. In the present embodiment of the PFB invention, the user can also directly enter (e.g. type) in the filter in editable box 1206. Furthermore, in the embodiment of Figure 10, the filter name may be any sequence of ASCII characters other than quotation marks. The filter name is limited to 32 characters in the present embodiment. However, the present invention is also well to allowing the filter to name to be restricted to fewer or greater than 32 characters. Also, the Verify button 1208 is used for syntax checking of the filter. In the present embodiment of the PFB invention, if errors are found the cursor is moved to the place where error is found. Furthermore, in the present embodiment, when the user clicks the OK button 1210, validation of the filters is performed. Thus, even in the more advanced embodiment, the present PFB invention allows a network manager or other packet filter builder to construct a packet filter without being extensively trained in or cognizant of all of the various parameters in packet filter construction.

With reference to Figure 11,a schematic diagram of the architecture of the implementation of Filter Parsing and Conversion utilized in accordance with the present claimed invention is shown.

With reference to Figure 12, a schematic diagram of the main classes and methods used in Filter Parsing employed in accordance with the present claimed invention is shown.

With reference now to Figures 13A-13C, flowcharts of the Operator Precedence Parsing Algorithm used in accordance with the present invention are shown. These algorithms are used for expressions containing operators except logical AND, logical OR. The main advantage of this algorithm is its simplicity & efficiency. No need to use recursive-descent method.

Referring now to Figure 14, an example of pseudocode utilized in accordance with the present claimed invention is shown. The pseudocode demonstrates Predictive Parsing Method. A predictive parser is a program consisting of a procedure for every nonterminal. Each procedure does two things: (i) it decides which production to use by looking at the lookahead symbol, and (ii) the procedure uses a production by mimicking the right side. A nonterminal results in a call to the procedure for the nonterminal, and a token matching the lookahead symbol results in the next input token being read. If at some point the token in the production does not match the lookahead symbol, an error is declared.

With reference next to Figure 15, an Expression-based Language Syntax Definition in BNF utilized in accordance with the present claimed invention.

The present PFB invention is also well suited to automatically validating the syntax of the user-defined packet filters. That is, the present PFB invention checks the constructed filters against a syntax diagram to insure that the packet filters are valid.

Similarly, the present PFB invention is also adapted to optimize the user-defined packet filters. That is, some packet filters can be written/built in many different ways. The present PFB invention is, however, adapted to analyze the constructed packet filters and optimize the structure thereof.

Thus, the present invention provides a method and computer system for packet filter building wherein the packet filter not constructed solely using complex design languages. The present invention further provides a packet filter building system and method which is not error-prone. Furthermore, the present invention provides a packet filter building system and method which does not require a highly trained programmer for the implementation thereof, and a packet filter building system and method which can be effectively and efficiently utilized by a typical network manager.

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order best to explain the principles of the invention and its practical application, to thereby enable others skilled in the art best to utilize the invention and various embodiments with various modifications suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.

# CLAIMS

What is Claimed is:

1. A computer implemented method of building a packet filter for a networked system, said computer implemented method comprising the step of:

a) generating a graphical user interface, said graphical user interface adapted to assist a user in the construction of a packet filter such that said user is able to build packet filter without being extensively trained in numerous parameters involved in packet filter construction.

2. The computer implemented method as recited in Claim 1 further comprising the steps of:

b) receiving from said user expression-based statements indicating characteristics of said packet filter;

c) converting said expression-based statements to a language suitable for the formation of said packet filter; and

d) creating said packet filter.

3. The computer implemented method as recited in Claim 1 further comprising the step of:

exporting said packet filter to a file.

4. The computer implemented method as recited in Claim 1 further comprising the step of:

loading said packet filter onto said networked system.

5. The computer implemented method as recited in Claim 1 wherein said graphical user interface is comprised of a wizard-type graphical user interface.

6. The computer implemented method as recited in Claim 1 wherein said graphical user interface is comprised of a calculator-type graphical user interface.

7. A computer implemented method of building a packet filter for a networked system, said computer implemented method comprising the steps of:

a) generating a graphical user interface, said graphical user interface adapted to assist a user in the construction of a packet filter; said graphical user interface adapted to assist said user in the following packet filter formation steps:

i) determining whether said packet filter involves first parameters;

ii) determining whether said packet filter involves second parameters; and

iii) constructing said packet filter.

8. The computer implemented method as described in step i) of Claim 7 wherein said graphical user interface assists said user in determining whether said packet filter involves address groups.

9. The computer implemented method as described in step ii) of Claim 7 wherein said graphical user interface assists said user in determining whether said packet filter involves port groups.

10. The computer implemented method as recited in Claim 8 wherein said graphical user interface assists said user in defining an address group for said packet filter when said packet filter involves address groups.

11. The computer implemented method as recited in Claim 9 wherein said graphical user interface assists said user in defining a port group for said packet filter when said packet filter involves port groups.

12. The computer implemented method as described in step iii) of Claim 7 wherein said graphical user interface assists said user in constructing said packet filter using an expression-based statement.

13. The computer implemented method as recited in Claim 7 wherein said graphical user interface is comprised of a wizard-type graphical user interface.

14. The computer implemented method as recited in Claim 7 wherein said graphical user interface is comprised of a calculator-type graphical user interface.

15. The computer implemented method as recited in Claim 7 wherein said graphical user interface is further adapted to assist said user in exporting said packet filter to a file.

16. The computer implemented method as recited in Claim 7 wherein said graphical user interface is further adapted to assist said user in loading said packet filter onto said networked system.

17. In a computer system having a processor coupled to a bus, a computer readable medium coupled to said bus and having stored therein a computer program that when executed by said processor causes said computer system to implement a method of assisting a user in the building of a packet filter for a networked system, said method comprising the step of:

a) generating a graphical user interface, said graphical user interface adapted to assist a user in the construction of a packet filter such that said user is able to build packet filter without being extensively trained in or cognizant of all of numerous parameters involved in packet filter construction.

18. A computer readable memory unit as described in Claim 17 wherein said computer implemented method stored on said computer readable medium further comprises the steps of:

b) receiving from said user expression-based statements indicating characteristics of said packet filter;

c) converting said expression-based statements to a language suitable for the formation of said packet filter; and

d) creating said packet filter.

19. The computer readable memory unit as described in Claim 17 wherein said computer implemented method stored on said computer readable medium further comprises the step of:

exporting said packet filter to a file.

20. The computer readable memory unit as described in Claim 17 wherein said computer implemented method stored on said computer readable medium further comprises the step of

loading said packet filter onto said networked system.

21. The computer readable memory unit as described in Claim 17 wherein said computer implemented method stored on said computer readable medium further comprises the step of:

generating a wizard-type graphical user interface to assist said user in the construction of said packet filter.

22. The computer readable memory unit as described in Claim 17 wherein said computer implemented method stored on said computer readable medium further comprises the step of:

generating a calculator-type graphical user interface to assist said user in the construction of said packet filter.

23. A method of building a packet filter, substantially as described herein, with reference to and as illustrated in the accompanying drawings.

The Patent Office

INVESTOR IN PEOPLE

**Application No:** GB 9821777.1
**Claims searched:** 1 to 16 and 23

**Examiner:** Grant Bedford
**Date of search:** 15 March 1999

## Patents Act 1977
## Search Report under Section 17

**Databases searched:**

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.Q): G4A (AKS APL AUB)

Int Cl (Ed.6): G06F (9/44 9/45 17/50)

Other: Online: EPODOC INTERNET PAJ WPI

**Documents considered to be relevant:**

| Category | Identity of document and relevant passage | Relevant to claims |
|---|---|---|
| X | Picking out packets for perusal, Review of EtherPeek at www.zdnet.com, 6 January 1997 | 1 to 16 and 23 |
| X | Review of Eagle 3.1 at www.zdnet.com, taken from ZD Internet Magazine February 1997 | 1 to 16 and 23 |
| X | NetSense ProAnalyst review at www.softseek.com and at www.net3group.com, 8 July 1997 | 1 to 16 and 23 |
| X,P | Anasil review at www.softseek.com and at www.lfnetworks.com 30 June 1998 | 1 to 16 and 23 |



| | | | |
|---|---|---|---|
| X | Document indicating lack of novelty or inventive step | A | Document indicating technological background and/or state of the art. |
| Y | Document indicating lack of inventive step if combined with one or more other documents of same category. | P | Document published on or after the declared priority date but before the filing date of this invention. |
| & | Member of the same patent family | E | Patent document published on or after, but with priority date earlier than, the filing date of this application. |